40장 이벤트

40.1 이벤트 드리븐 프로그래밍

> **이벤트 드리븐 프로그래밍** : 프로그램의 흐름을 **이벤트 중심**으로 제어하는 프로그래밍 방식

- 브라우저는 처리해야 할 특정 사건이 발생하면 **이벤트** 발생
  - 클릭, 키보드 입력, 마우스 이동 등등...
- 이벤트 핸들러 : 이벤트가 발생했을 때 호출될 **함수**
- 이벤트 핸들러 등록 : 이벤트 발생 시 브라우저에게 이벤트 핸들러의 호출 **위임**

40.2 이벤트 타입

40.2.1 마우스 이벤트

이벤트 타입	이벤트 발생 시점
click	마우스 버튼을 클릭했을 때
dblclick	마우스 버튼을 더블 클릭했을 때
mousedown	마우스 버튼을 눌렀을 때
mouseup	누르고 있던 마우스 버튼을 놓았을 때
mousemove	마우스 커서를 움직였을 때
mouseenter	마우스 커서를 HTML 요소 안으로 이동했을 때(버블링[2]되지 않는다)
mouseover	마우스 커서를 HTML 요소 안으로 이동했을 때(버블링된다)

이벤트 타입	이벤트 발생 시점
mouseleave	마우스 커서를 HTML 요소 밖으로 이동했을 때(버블링되지 않는다)
mouseout	마우스 커서를 HTML 요소 밖으로 이동했을 때(버블링된다)

40.2.2 키보드 이벤트

이벤트 타입	이벤트 발생 시점
keydown	모든 키를 눌렀을 때 발생한다. ※ control, option, shift, tab, delete, enter, 방향 키와 문자, 숫자, 특수 문자 키를 눌렀을 때 발생한다. 단, 문자, 숫자, 특수 문자, enter 키를 눌렀을 때는 연속적으로 발생하지만 그 외의 키를 눌렸을 때는 한 번만 발생한다.
keypress	문자 키를 눌렀을 때 연속적으로 발생한다. ※ control, option, shift, tab, delete, 방향 키 등을 눌렀을 때는 발생하지 않고 문자, 숫자, 특수 문자, enter키를 눌렀을 때만 발생한다. 폐지(deprecated)되었으므로 사용하지 않을 것을 권장한다.
keyup	누르고 있던 키를 놓았을 때 한 번만 발생한다. ※ keydown 이벤트와 마찬가지로 control, option, shift, tab, delete, enter, 방향 키와 문자, 숫자, 특수 문자 키를 놓았을 때 발생한다.

40.2.3 포커스 이벤트

이벤트 타입	이벤트 발생 시점
focus	HTML 요소가 포커스를 받았을 때(버블링되지 않는다)
blur	HTML 요소가 포커스를 잃었을 때(버블링되지 않는다)
focusin	HTML 요소가 포커스를 받았을 때(버블링된다)
focusout	HTML 요소가 포커스를 잃었을 때(버블링된다)

40.2.4 폼 이벤트

이벤트 타입	이벤트 발생 시점
submit	1. form 요소 내의 input(text, checkbox, radio), select 입력 필드(textarea 제외)에서 엔터 키를 눌렀을 때 2. form 요소 내의 submit 버튼(<button>, <input type="submit">)을 클릭했을 때 ※ submit 이벤트는 form 요소에서 발생한다.

40.2.5 값 변경 이벤트

이벤트 타입	이벤트 발생 시점
input	input(text, checkbox, radio), select, textarea 요소의 값이 입력되었을 때
change	input(text, checkbox, radio), select, textarea 요소의 값이 변경되었을 때 ※ change 이벤트는 input 이벤트와는 달리 HTML 요소가 포커스를 잃었을 때 사용자 입력이 종료되었다고 인식하여 발생한다. 즉, 사용자가 입력을 하고 있을 때는 input 이벤트가 발생하고 사용자 입력이 종료되어 값이 변경되면 change 이벤트가 발생한다.
readystatechange	HTML 문서의 로드와 파싱 상태를 나타내는 document.readyState 프로퍼티 값('loading', 'interactive', 'complete')이 변경될 때

40.2.6 DOM 뮤테이션 이벤트

이벤트 타입	이벤트 발생 시점
DOMContentLoaded	HTML 문서의 로드와 파싱이 완료되어 DOM 생성이 완료되었을 때

40.2.7 뷰 이벤트

이벤트 타입	이벤트 발생 시점
resize	브라우저 윈도우(window)의 크기를 리사이즈할 때 연속적으로 발생한다. ※ 오직 window 객체에서만 발생한다.
scroll	웹페이지(document) 또는 HTML 요소를 스크롤할 때 연속적으로 발생한다.

40.2.8 리소스 이벤트

이벤트 타입	이벤트 발생 시점
load	DOMContentLoaded 이벤트가 발생한 이후, 모든 리소스(이미지, 폰트 등)의 로딩이 완료되었을 때(주로 window 객체에서 발생)
unload	리소스가 언로드될 때(주로 새로운 웹페이지를 요청한 경우)
abort	리소스 로딩이 중단되었을 때
error	리소스 로딩이 실패했을 때

40.3 이벤트 핸들러 등록

40.3.1 이벤트 핸들러 어트리뷰트 방식

- on 접두사 + 이벤트 타입
  - ex) onclick, onchange 등등..
- 이벤트 핸들러 어트리뷰트 값은 함수 참조가 아닌 함수 호출문 등의 **문**
- 원래는 함수가 아닌 값을 반환하는 함수 호출문을 등록하면 호출 불가하나,
  암묵적으로 생성될 이벤트 핸들러의 함수가 됨.

```
<button onclick="sayHi('Lee')">click</button>

function onclick(event) {
    sayHi('Lee');
}

<button onclick="sayHi">click</button>
// 매개변수 없는 함수 참조를 할당하면 인수를 전달할 수 없다..
```

40.3.2 이벤트 핸들러 프로퍼티 방식

- 어트리뷰트 방식처럼 on 접두사를 사용하지만 이벤트 핸들러 프로퍼티에 이벤트 핸들러를 바인딩

```
<button>Click</button>
<script>
    const $button = document.querySelector('button');

    // 이벤트 핸들러 프로퍼티에 이벤트 핸들러를 바인딩
    $button.onclick = function () {
        console.log('button click');
    };
</script>
```

그림 40–2 이벤트 핸들러 프로퍼티 방식

- 이벤트 핸들러는 이벤트 타깃 혹은 전파된 이벤트를 캐치할 DOM 노드 객체에 바인딩
- 이벤트 핸들러 어트리뷰트 방식의 특징인 HTML과 자바스크립트가 뒤섞이는 문제를 해결
- 그러나 이벤트 핸들러 프로퍼티에 하나의 이벤트 핸들러만 바인딩 가능(재할당)

40.3.3 addEventListener 메서드 방식

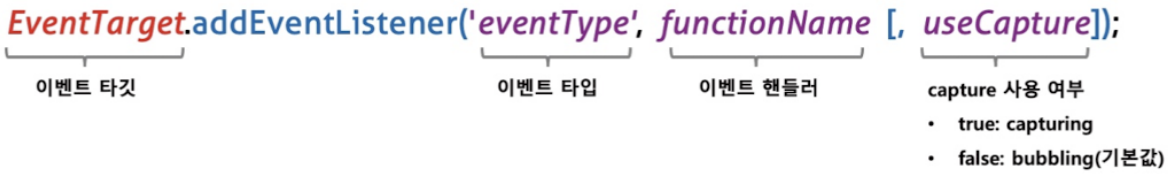

그림 40–3 addEventListener 메서드

```
<button>Click</button>
<script>
    const $button = document.querySelector('button');

    // 이벤트 핸들러 프로퍼티에 이벤트 핸들러를 바인딩
    // $button.onclick = function () {
    //   console.log('button click');
    // };

    // addEventListener 메서드 방식
```

```
    $button.addEventListener('click', function () {
      console.log(;button click');
    });
</script>
```

- 프로퍼티 방식과 메서드 방식은 서로 별개의 방식이기에 모두 호출 가능
- 메서드 방식은 하나 이상의 이벤트 핸들러 등록 가능
  - 이때 등록된 순서대로 호출
- 참조가 동일한 이벤트 핸들러 중복 등록 시 하나만 등록

40.4 이벤트 핸들러 제거

- `EventTarget.prototype.removeEventListener` 메서드 사용
- 삭제 시 등록한 인수와 정확히 동일해야 함
- 무명 함수는 참조 불가하기에 제거 불가
- 기명 이벤트 핸들러 내부에서 removeEventListener 사용 가능
  - 무명 이벤트 핸들러일 경우 arguments.callee 사용하여 제거 가능
    - 코드 최적화에 방해되기에 strict mode 사용 금지
- 프로퍼티 방식의 핸들러 삭제는 removeEventListener 메서드로 제거 불가
  - 이벤트 핸들러 프로퍼티에 null 할당하는 방식으로 대체

40.5 이벤트 객체

```
<!DOCTYPE html>
<html>
<body>
    <p>클릭하세요. 클릭한 곳의 좌표가 표시됩니다.</p>
    <em class="message"></em>
        <script>
      const $msg = document.querySelector('.message');
```

```
            // 클릭 이벤트에 의해 생성된 이벤트 객체는 이벤트 핸들러
의 첫 번째 인수로 전달
      function showCoords(e){
        $msg.textContent = `clientX: ${e.clientX}, clientY:
${e.clientY}`;
      }

      document.onclick = showCoords;
    </script>
</body>
</html>
```

- 이벤트 핸들러 어트리뷰트 방식은 첫 번째 매개변수 이름이 event여야 함
  - 암묵적으로 생성 및 파싱되어 자동으로 매개변수가 event로 명명됨

40.5.1 이벤트 객체의 상속 구조

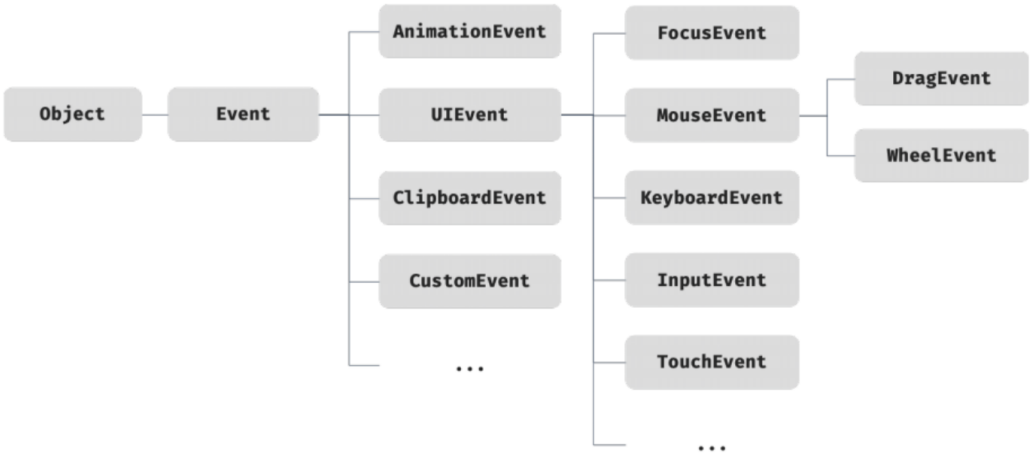

그림 40-4 이벤트 객체의 상속 구조

```
<script>
    // Event 생성자 함수를 호출하여 foo 이벤트 타입의 Event 객체를 생성
    let e = new Event('foo');
</script>
```

40.5.2 이벤트 객체의 공통 프로퍼티

공통 프로퍼티	설명	타입
type	이벤트 타입	string
target	이벤트를 발생시킨 DOM 요소	DOM 요소 노드
currentTarget	이벤트 핸들러가 바인딩된 DOM 요소	DOM 요소 노드
eventPhase	이벤트 전파 단계 0: 이벤트 없음, 1: 캡처링 단계, 2: 타깃 단계, 3: 버블링 단계	number
bubbles	이벤트를 버블링으로 전파하는지 여부. 다음 이벤트는 bubbles: false로 버블링하지 않는다. ▪ 포커스 이벤트 focus/blur ▪ 리소스 이벤트 load/unload/abort/error ▪ 마우스 이벤트 mouseenter/mouseleave	boolean
cancelable	preventDefault 메서드를 호출하여 이벤트의 기본 동작을 취소할 수 있는지 여부. 다음 이벤트는 cancelable: false로 취소할 수 없다. ▪ 포커스 이벤트 focus/blur ▪ 리소스 이벤트 load/unload/abort/error ▪ 마우스 이벤트 dblclick/mouseenter/mouseleave	boolean

공통 프로퍼티	설명	타입
defaultPrevented	preventDefault 메서드를 호출하여 이벤트를 취소했는지 여부	boolean
isTrusted	사용자의 행위에 의해 발생한 이벤트인지 여부. 예를 들어, click 메서드 또는 dispatchEvent 메서드를 통해 인위적으로 발생시킨 이벤트인 경우 isTrusted는 false다.[8]	boolean
timeStamp	이벤트가 발생한 시각(1970/01/01/00:00:0부터 경과한 밀리초)	number

40.5.3 마우스 정보 취득

- 마우스 포인터의 좌표 정보를 나타내는 프로퍼티
  - screenX/screenY, clientX/clientY, pageX/pageY, offsetX/offsetY
- 버튼 정보를 나타내는 프로퍼티
  - altKey, ctrlKey, button

40.5.4 키보드 정보 취득

- keydown, keyup, keypress 이벤트 발생 시 생성되는 KeyboardEvent 타입의 이벤트 객체
    - altKey, ctrlKey, shiftKey, key, metaKey, keyCode 등(keyCode 폐지, key 프로퍼티 권장)
- 입력한 키와 key 프로퍼티 값의 대응 관계
    - https://keycode.info
- input 요소의 입력 필드에 한글 입력 후 엔터 키를 누르면 keyup 이벤트 헨들러가 두 번 호출되는 현상
    - keyup 이벤트 대신 keydown 사용

40.6 이벤트 전파

- **이벤트 전파**
    - DOM 트리 상에 존재하는 DOM 요소 노드에서 발생한 이벤트는 DOM 트리를 통해 전파

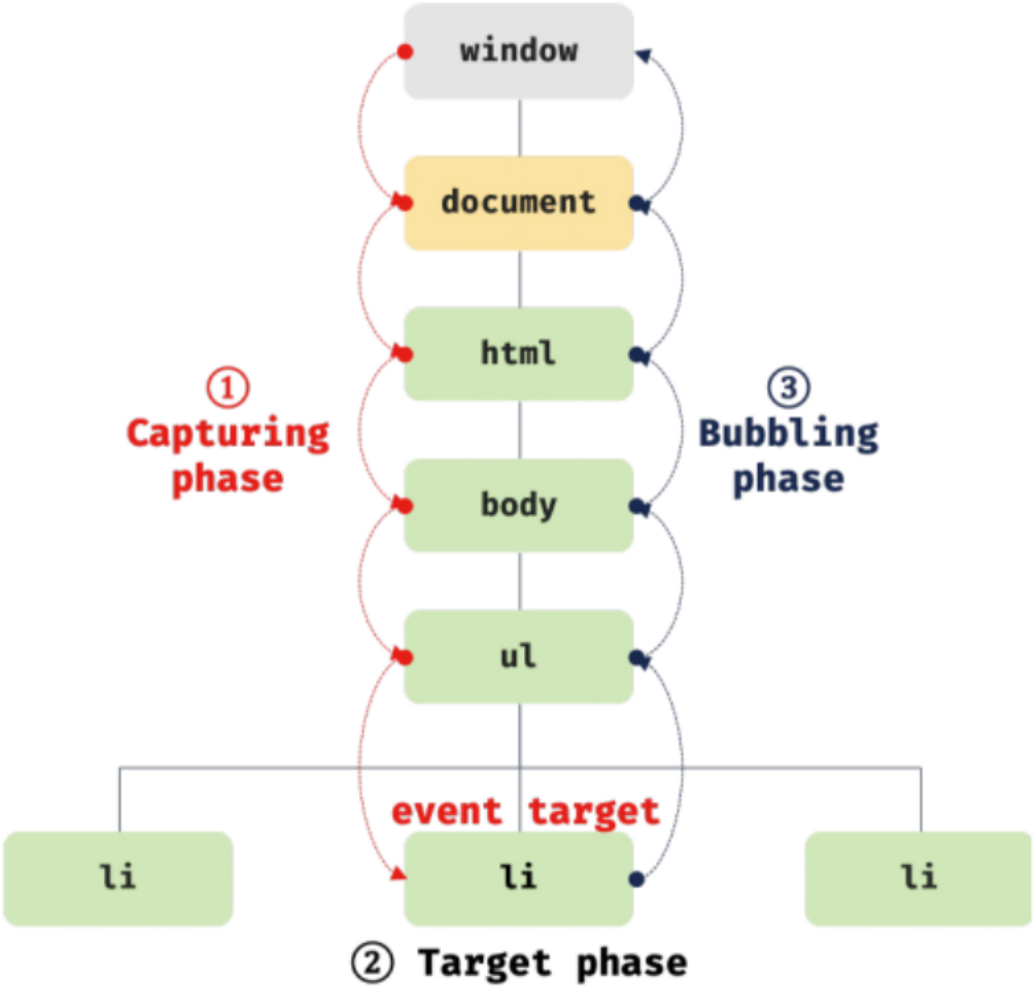

그림 40-8 이벤트 전파

1. 캡처링 단계 : 이벤트가 상위 요소에서 하위 요소로 전파
2. 타깃 단계 : 이벤트가 이벤트 타깃에 도달
3. 버블링 단계 : 이벤트가 하위 요소에서 상위 요소 방향으로 전

```
<ul id="fruits">
    <li>Apple</li>
    <li>Apple</li>
    <li>Apple</li>
</ul>
<script>
    const $fruits = document.getElementById('fruits');
```

```
    // #fruit 요소의 하위 요소인 li 요소를 클릭한 경우
    $fruits.addEventListener('click', e => {
        console.log(`이벤트 단계: ${e.eventPhase}`); // 3: 버
블링 단계
        console.log(`이벤트 타깃: ${e.target}`); // [object H
TMLElement]
        console.log(`커린트 타깃: ${e.currentTarger}`); // [o
bject HTMLElement]
    });
</script>
```

1. li 요소 클릭하여 이벤트 타깃이 된 후 'click' 이벤트 객체는 window에서 시작해서 이벤트 타깃 방향으로 전파

   ⇒ 캡처링 단계

2. 'click' 이벤트 객체는 이벤트를 발생시킨 이벤트 타깃(li 요소)에 도달

   ⇒ 타깃 단계

3. 'click' 이벤트 객체는 이벤트 타깃(li 요소)에서 시작해서 window 방향으로 전파

   ⇒ 버블링 단계

- 어트리뷰트/프로퍼티 방식의 핸들러는 타깃 단계와 버블링 단계만 캐치
- 메서드 방식은 캡처링 단계까지 캐치 가능
  - 이때는 3번째 인수로 true 전달해야 가능
- 일부 이벤트는 event.bubbles의 값이 false이기에 버블링을 통해 전파되지 않음
  - 캡처링 단계에서 캐치하거나, 상위 요소에서 캐치해야 한다면 대체 이벤트를 사용

40.7 이벤트 위임

- **이벤트 위임** : 여러 개의 하위 DOM 요소에 각각 이벤트 핸들러를 등록하는 대신 **하나의 상위 DOM 요소**에 이벤트 핸들러를 등록하는 방법
  - 여러 개의 하위 DOM 요소에 이벤트 핸들러를 등록할 필요 없음
  - 동적으로 하위 DOM 요소가 추가되더라도 일일히 등록할 필요 없음

- !! 이벤트에 반응이 필요한 DOM 요소에 한정하여 이벤트 핸들러가 실행되도록 검사할 필요 있음
  - `Element.prototype.matches` 메서드 사용

```
// 이벤트 위임
document.getElementById('fruits).onclick = function () {
    if (!target.matches('#fruits > li')) return;
    // ~~~
}

// 각 요소에 직접 이벤트 리스너 추가하는 방식
document.querySelectorAll('#fruits li').forEach(function(e
    element.addEventListener('click', function() {
        // ~~~
  });
});

// jquery g..
$('#fruits li').on('click', function() {
    // ~~~
});
```

💡 **각 요소에 이벤트 리스너 추가 VS 이벤트 위임**
1. 각~
- 🙂명확하고 직관적
- 😡동적 요소 발생 시 추가된 요소에는 적용 X

2. 이벤트 위임
- 🙂성능 최적화(하나의 이벤트 리스너만 추가)
- 🙂동적 요소 처리
- 😡특정 요소 증가마다 추가 조건문 증가

⇒ 지정 요소가 적을 땐 각~, 동적 요소 발생할 땐 이벤트 위임

40.8 DOM 요소의 기본 동작 조작

40.8.1 DOM 요소의 기본 동작 중단

- a 요소를 클릭하면 href 어트리뷰트에 지정된 링크로 이동하는 등 저마다 기본 동작이 존재
- `preventDefault` 메서드는 이러한 DOM 요소의 기본 동작을 **중단**
  - 주로 form 제출 시 페이지 새로 고침 방지로 많이 쓰는 듯..?
  - 이 외에 키보드 이벤트 기본 동작 방지 등에 사용해도 좋을 듯

40.8.2 이벤트 전파 방지

- `stopProgagation` 메서드 : 이벤트 전파 중지
  - 상위 DOM으로 전파되는 것을 중단하고 자신의 DOM 요소에서만 개별적으로 이벤트 처리

40.9 이벤트 핸들러 내부의 this

40.9.1 이벤트 핸들러 어트리뷰트 방식

- 이벤트 핸들러에 의해 일반 함수로 호출되고, 이는 전역 객체를 가리킴
- 이벤트 핸들러 호출 시 인수로 전달한 this는 이벤트를 바인딩한 DOM 요소를 가리킴

40.9.2 이벤트 핸들러 프로퍼티 방식과 addEventListener 메서드 방식

- 모두 이벤트를 바인딩한 DOM 요소를 가리킴
  - 이벤트 핸들러 내부의 this = 이벤트 객체의 currentTargert 프로퍼티
- 화살표 함수로 정의한 이벤트 핸들러 내부 this는 상위 스코프의 this
  - 화살표 함수는 함수 자체의 this 바인딩이 없기 때문

40.10 이벤트 핸들러에 인수 전달

- 이벤트 핸들러 프로퍼티 방식과 메서드 방식
  - 이벤트 핸들러의 **내부**에서 함수를 호출하면서 인수 전달
  - 이벤트 핸들러를 **반환**하는 함수를 호출하면서 인수 전달

40.11 커스텀 이벤트

40.11.1 커스텀 이벤트 생성

- CustomEvent 생성자 함수로 새로운 이벤트 타입 지정
- 버블링되지 않으며, preventDefault 메서드로 취소 불가능
  - bubbles와 cancelable 프로퍼티의 값이 false로 기본 설정
  - true로 설정하려면 생성자 함수의 두 번째 인수로 해당 프로퍼티를 갖는 객체 전달
- 이벤트 생성자 함수로 생성한 커스텀 이벤트는 isTrusted 프로퍼티 값 false
- 사용자의 행위로 발생한 이벤트에 의해 생성된 이벤트 객체의 isTrusted 프로퍼티 값 true

40.11.2 커스텀 이벤트 디스패치

- `dispatchEvent` 메서드
  - disoatchEvent 메서드에 이벤트 객체를 인수로 전달 및 호출하면 인수로 전달한 이벤트 타입의 이벤트 발생
- 일반적인 이벤트 핸들러는 동기 방식인데 반해 dispatchEvent 메서드는 동기 방식

> 💡 커스텀 이벤트 객체 생성 시 addEventListener 메서드 방식으로 이벤트 핸들러 등록
> ⇒ on으로 시작하는 어트리뷰트/프로퍼티가 요소 노드에 존재하지 않기 때문